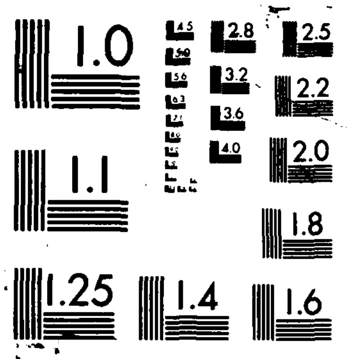
AD-A182 240 EXACT PERFORMANCE ANALYSIS OF TWO DISTRIBUTED PROCESSES WITH ONE SYNCHRON (U) MARYLAND UNIV COLLEGE PARK DEPT OF COMPUTER SCIENCE M ABRAMS ET AL MAY 87 CS-TR-1844 1/1

UNCLASSIFIED N00014-87-K-0124 F/G 12/7 NL

END
8-87
DTIC



CROCOPY RESOLUTION TEST CHART

REPORT DOCUMENTATION PAGE

AD-A182 240

1b. RESTRICTIVE MARKINGS N/A	
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE N/A	3. DISTRIBUTION / AVAILABILITY OF REPORT approved for public release; distribution unlimited.
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CS-TR-1844	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION University of Maryland; 6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION Office of Naval Research
6c. ADDRESS (City, State, and ZIP Code) Dept. of Computer Science, University of Maryland, College Park, MD 20742	7b. ADDRESS (City, State, and ZIP Code) 800 North Quincy Street, Arlington, VA 22217-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-87-K-0124
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Exact Performance Analysis of Two Distributed Processes with One Synchronization Point

12. PERSONAL AUTHOR(S)
Marc Abrams and Ashok K. Agrawala

13a. TYPE OF REPORT Technical	13b. TIME COVERED N/A FROM ____ TO ____	14. DATE OF REPORT (Year, Month, Day) May 1987	15. PAGE COUNT 34

16. SUPPLEMENTARY NOTATION

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
[illegible]

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

~~We~~ seek fundamental insight into the role synchronization plays in distributed programs. Therefore, ~~we~~ study the execution behavior of two cyclic processes synchronizing once per cycle. This occurs when two processes unilaterally share a resource. ~~We~~ assume the execution time of each process in isolation is known and deterministic. Through a Diophantine equation (whose coefficients and unknowns are integers), ~~we~~ obtain the time one process waits to use the resource as a function of how long the other process uses the resource. This function is found to be piece-wise constant. A complex expression for the location of discontinuities is found. Implications of the results on writing distributed programs are discussed.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

CS-TR-1844 May 1987

EXACT PERFORMANCE ANALYSIS OF TWO DISTRIBUTED PROCESSES WITH ONE SYNCHRONIZATION POINT*

Marc Abrams and Ashok K. Agrawala

COMPUTER SCIENCE
TECHNICAL REPORT SERIES

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND
20742

CS-TR-1844 May 1987

EXACT PERFORMANCE ANALYSIS OF TWO DISTRIBUTED PROCESSES WITH ONE SYNCHRONIZATION POINT[*]

Marc Abrams and Ashok K. Agrawala

Department of Computer Science
University of Maryland
College Park MD 20742

(*) This research was supported in part by a contract (N00014-87-K-0241) from The Office of Naval Research to The Department of Computer Science University of Maryland

Exact Performance Analysis of Two Distributed Processes with One Synchronization Point

MARC ABRAMS and ASHOK K. AGRAWALA

University of Maryland

We seek fundamental insight into the role synchronization plays in distributed programs. Therefore, we study the execution behavior of two cyclic processes synchronizing once per cycle. This occurs when two processes unilaterally share a resource. We assume the execution time of each process in isolation is known and deterministic. Through a Diophantine equation (whose coefficients and unknowns are integers), we obtain the time one process waits to use the resource as a function of how long the other process uses the resource. This function is found to be piece-wise constant. A complex expression for the location of discontinuities is found. Implications of the results on writing distributed programs are discussed.

Categories and Subject Descriptors: D.2.8 [**Software**]: Metrics – *Performance measures;* D.4.8 [**Operating Systems**]: Performance – *Modeling and prediction,* C.4 [**Performance of Systems**]: Modeling techniques

General Terms: Distributed Programs, Performance, Synchronization

Additional Key Words and Phrases: Resource sharing, Diophantine equation

This research was supported by Grant NSF-DCR-8405235 from the National Science Foundation.

Authors' current addresses: M. Abrams, IBM Research Division, Zurich Research Laboratory, 8803 Rüschlikon, Switzerland; A.K. Agrawala, Department of Computer Science, University of Maryland, College Park, MD 20742.

November 27, 1986

1. INTRODUCTION

A distributed computer program is often a collection of asynchronous, sequential processes that occasionally synchronize or communicate. One trend in modeling such programs is to develop techniques powerful enough to model arbitrarily large programs. Queueing network models [5] and performance Petri nets [6] are principal techniques.

A complementary approach is to model small, elementary programs in exchange for the ability to perform the analysis exactly. Analytic formulae describing the exact run-time behavior may give fundamental insights into how synchronization affects distributed program performance. Such is the approach of this paper. We start by explaining the elementary type of program studied here.

2. PROGRAM MODELED

Consider a program consisting of two processes, each of which is executed by an asynchronous, dedicated processor. Both processes share a single resource, in a manner to be described shortly. Each process repeats forever the following sequence of two *local states*:

(1) perform operations not requiring the resource (state 0);

(2) perform operations requiring the resource (state 1).[1]

[1] The state name denotes the number of resources the process is using.

The length of time the process spends in each state is a known, positive, real quantity. The sum of the time process 1 spends in states 0 and 1 is denoted c_1.[2] We define c_2 for process 2 similarly. The time process 2 uses the resource, i.e., the time of state 1, is denoted by $l \in (0, c_2)$. Variables c_1, c_2, and l are real quantities.

One way in which resources *could* be shared is *mutually exclusively*: neither process enters state 1 and uses the resource, when the other process is in state 1. This case is analyzed in [1], and requires a *numerical algorithm* to obtain the solution. The algorithm permits only an empirical understanding of the solution.

In this paper, we analyze the more restrictive case of *unilaterally-exclusive* resource sharing to obtain a *closed-form* solution.[3] The closed form exposes the functional dependence of behavior on parameters c_1, c_2, and l. In unilateral resource sharing, only one of the processes obeys the protocol of not entering state 1 when the other process is already in state 1. However, unlike mutually-exclusive sharing, the other process ignores the protocol and uses the resource whenever it wishes. Let process 1 obey the protocol, and process 2 use the resource at will.

Two unilaterally-exclusive resource users together produce mutually-exclusive resource use. Thus, the analysis of unilaterally-exclusive resource use is a fundamental problem.

[2] Quantity c_1 may also be thought of as the time required to execute one cycle of process 1 excluding the time spent waiting for the resource.

[3] The solution obtained is in closed form to the extent that the expression contains a parameter which is the solution of a linear congruence, and closed-form solutions to congruences are not known.

3. ANALYSIS PROBLEMS

Each time process 1 completes execution of state 0, it may need to wait before it can use the resource, and thus enter state 1. The overall problem in this paper is to determine the functional dependence of the waiting time on the length of time process 2 uses the resource (i.e., l). This function is denoted $w(l)$; its domain and range are the real numbers. We shall show that the domain of $w(l)$, which is $[0, c_2)$, may be partitioned into a set of intervals. Function w is *constant* within each interval, but *discontinuous* at interval boundaries.

The *global state* of a program consists of the local state of all processes. We describe the execution behavior of a program by enumerating the sequence of

(1) global-state transitions occurring during execution, and

(2) *occupancy times*, or time spent in each global state.

A program is in *steady state* when it repeats a finite sequence of global states and occupancy times forever. The time required for each repetition of the sequence is called the *steady-state period*.

In this paper, we find for every program fitting our model

(1) a proof that every program reaches a steady state;

(2) the steady-state period;

(3) the necessary and sufficient conditions for a process to wait for a resource, given rational cycle times c_1 and c_2; and

(4) an expression for w(l), the time a process waits each time it uses the resource, based on four conjectures about the form of w.

Our chief analytic difficulty is that we only know the global state at the moment both processes start execution. The asynchronous nature of the processes precludes *a priori* knowledge of the global state at any later time. Therefore, we cannot apply results from scheduling theory [2], critical path analysis [2], the performance of flexible manufacturing systems [3], and related fields that analyze the time at which a deterministic sequence of state transitions occurs.

4. STEADY-STATE EQUATION

In this section, we analyze the steady-state program behavior. The two processes of a distributed program may be started at two different times. These two times form the *initial condition* of the program. First, we shall discuss why any program reaches steady state regardless of the initial condition. Then we derive the condition under which one process waits for the resource in steady state.

When a program starts execution from any initial condition, one of two mutually-exclusive cases occurs:[4]

CASE 1. Process 1 never waits for the resource.

CASE 2. Process 1 waits at least once for the resource.

---

[4] Note that process 2 never waits by definition of unilateral sharing.

Consider case 1 first. Because no process waits, process 1 repeats its state transitions every c_1 time units, and process 2 repeats every c_2 units. Therefore, the moment both processes have started, the program is in steady state. Further, the steady-state period is c_1c_2 time units.

Next, consider case 2. Without loss of generality, let time zero be the moment process 2 releases the resource (leaves state 1), thereby permitting process 1 to enter state 1. The processes progress in time as shown in Figure 1. Looking at the progress of process 2 in Figure 1, the resource is unavailable to process 1 during time intervals $[c_2 - l, c_2)$, $[2c_2 - l, 2c_2)$, $[3c_2 - l, 3c_2)$, ... Meanwhile, process 1 will start using the resource at times 0, c_1, $2c_1$, $3c_1$, ... *until* some time that is an integral multiple of c_1 and lies within the interval when process 2 uses the resource. Therefore, process 1 will wait for the resource if and only if

$$\exists i \in I,\ \exists j \in I \quad \ni \quad ic_1 \in [jc_2 - l,\ jc_2) \ , \qquad (1)$$

where I denotes the positive integers. The *smallest* i and j satisfying (1) represent the number of loops processes 1 and 2 make, respectively, before process 1 waits.

If (1) is satisfiable, the system will return at time jc_2 to the state it was in at time 0. Thus, its behavior will repeat, so that (1) represents the steady-state behavior. Alternately, if (1) is not satisfiable, a desirable situation will occur in which process 1 waits exactly once, after which the processes use the resource without conflict and without waiting. This is a steady-state behavior because the state at time 0 is returned to at time c_1c_2. Thus, we have:

Theorem 1. *A two-process distributed program in which process 1 unilaterally shares a resource with process 2 will always reach a steady state. Let the cycle times be c_1 and c_2. If waiting occurs, the steady-state period will be jc_2, where j is the smallest, positive integer satisfying (1). Otherwise, neither process waits for the resource, and the period is c_1c_2.*

Expression (1) is equivalent to the condition that process 1 will wait if and only if

$$\exists i \in I,\ \exists j \in I,\ \exists x \in (0,I] \ni \quad ic_1 - jc_2 + x = 0 \ . \tag{2}$$

Equation (2) will either have no solution or an infinite number of solutions [4]. However, the solution corresponding to the program behavior is the minimum positive integers i and j satisfying (2), for the following reason: The first time process 1 leaves state 0 and the resource is in use, process 1 will wait. This corresponds to the minimum number of loops each process makes before contending for the resource, and hence the minimum positive integers i and j satisfying (2).[5]

[5] Larger, positive values of i and j correspond to physically unrealizable situations, in which the first n (n > 0) times process 1 leaves state 0 and the resource is in use, process 1 illegally uses the resource without waiting.

By definition, $w(l)$ is the value of x which satisfies (2) when the minimum values of i and j are substituted.

Solving equation (2) presents three problems:

(1) obtaining the condition under which a solution exists;

(2) allowing only integer values for the solution of variables i and j, and;

(3) finding the *minimal* values of i and j satisfying (2).

The first two problems are solved below by transforming (2) to a Diophantine equation,[6] and applying known methods. However, the third problem is difficult. To our knowledge, finding minimum integer solutions analytically is not addressed in the literature. Solutions to these problems are presented next.

Earlier, we defined c_1 and c_2 to be *real* quantities. For the following analysis, we shall assume that c_1 and c_2 are rational quantities.[7] Henceforth, we shall equivalently assume that c_1 and c_2 are *relatively prime integers*.

To demonstrate the equivalence, we can multiply each term in (2) by the factor L/G, where L is the least common denominator of c_1 and c_2, and G is the greatest common divisor (g.c.d.) of c_1 and c_2. The resulting solution to i, j, and x in (2) must be multiplied by G. Finally, l must be divided by L.

[6] A Diophantine equation has integer coefficients and unknowns, as discussed in [4].

[7] The subsequent results do not apply to processes whose cycle time is irrational. Although this case is theoretically interesting, in practice we measure programs using only rational numbers (e.g., all measurements have a precision of one microsecond).

Because quantities i, j, c_1, and c_2 are integers, x must only have integers as its range. Thus, equation (2) may be considered to be a Diophantine equation [4].

Now, in the following theorem and corollary, we may answer a question of practical importance:

> How can we design a two-process, deterministic program to unilaterally share a resource *without conflict* in steady state?

We seek the condition under which the periods of resource use of the two processes are interleaved in such a way that neither process waits in steady state.

Theorem 2. *Consider a two-process distributed program in which process 1 unilaterally shares a resource with process 2, and whose cycle times* c_1 *and* c_2 *are rational quantities. At steady state, neither process will ever wait for the resource if and only if the duration for which process 2 uses the resource (i.e.,* l*) is less than the g.c.d. of* c_1 *and* c_2.

Proof. The definition of unilateral sharing only permits process 1 to wait. Process 1 waits if and only if Diophantine equation (2) has a solution. Theorem 3.3 in [4] gives the condition for solution:

> A necessary and sufficient condition for the Diophantine equation $a_1z_1 + a_2z_2 + \dots + a_nz_n = k$ to have a solution is that the greatest common divisor (g.c.d.) of the a_i's divides k.

Let c denote the g.c.d. of c_1 and c_2. If we consider for a moment x to be an integer constant k, then by the above theorem equation (2) has a solution if and only if c divides k. Now k, or equivalently x, assumes any integer in the range $(0, l]$. Thus, equation (2) has a solution if and only if $l \geq c$. □

Corollary. *If a two-process distributed program unilaterally sharing a resource has equal cycle lengths ($c_1 = c_2$), then the steady state never requires either process to wait for the resource.*

Proof. Neither process waits if and only if equation (2) has no solution. A solution exists if and only if the g.c.d. of c_1 and c_2, which is c_2, divides some integer in the range $(0,l]$. Because the time a process spends in state 0 and 1 must be positive, $l < c_2$. Thus, no such integer exists. □

Next, we find *all* solutions to (2). Using the technique described in [4], page 66-68, the infinite number of solutions to (2) are

$$i = c_2 y + tx \tag{3a}$$

$$j = c_1 y + rx \ , \tag{3b}$$

where y can assume any integer value and r and t are any integers satisfying[8]

$$c_2 r - c_1 t = 1 \ . \tag{3c}$$

[8] A solution to (3c) may be found by solving the following linear congruence by known methods $c_2 r \equiv 1 \pmod{c_1}$.

The value of x that minimizes i will also be the value minimizing j (refer to Figure 1). Henceforth, we need only analyze (3a). Parameter y may be eliminated by using conditions $i \geq 0$ and $j \geq 0$. Combining these with (3a) and (3b) yields

$$y \geq \max\left(\frac{-tx}{c_2}, \frac{-rx}{c_1}\right) .$$

From (3c), the first term in the maximum is larger. Using the notation $\lfloor x \rfloor$ to represent the largest integer that is less than or equal to x, and substituting into (3a) yields

$$i = tx - c_2 \left\lfloor \frac{tx}{c_2} \right\rfloor . \tag{4}$$

This is equivalent to

$$i = (tx \mod c_2) . \tag{5}$$

5. EXPRESSION FOR FUNCTION w(l)

To find the steady-state waiting time w(l), we started in equation (2) by seeking the minimum number of loops each process makes before resource contention, denoted i and j. Knowing these, w(l) is the corresponding resource waiting time x. We transformed this problem into (5), which says that $w(l)$ is the value $x \in (0,l]$ minimizing i. This minimum is investigated below to develop an expression for $w(l)$.

First, observe that $w(l)$ is a step function. To see this, suppose the value of l is the constant l_0. Then the value $w(l_0)$ is the integer in the range $(0, l_0]$ minimizing i in (5). If l is increased to $l_0 + \Delta l$, where Δl is a real number in $(0, c_2 - l_0)$, then $w(l_0 + \Delta l)$ may *still* be equal to $w(l_0)$. This occurs if and only if $(tx \bmod c_2) > (tw(l_0) \bmod c_2)$, $\forall x \in (l_0, l_0 + \Delta l]$. Thus, $w(l)$ is a step function.

The crucial problem is to determine the *critical points* of w, or those values of l at which w is discontinuous. A critical point is defined to be an integer $c \in (0,c_2]$ such that $(tc \bmod c_2) \leq (tz \bmod c_2)$, $\forall z = 1,2,\ldots,c$. We shall define 0 to be a critical point, as a convenience in our derivation. Thus, the critical points are the successive minima in a graph of i versus x, as we scan the graph from $x = 0$ to $x = c_2$.

Example. Despite its simple appea:ance, function $i = (tx \bmod c_2)$ generates a rich variety of functions, as illustrated in Figure 2. Henceforth, we only use the case of $c_1 = 29$, $c_2 = 50$ (Figure 2b) for illustrating the conjectures. An example of the function w is illustrated in Figure 3, corresponding to Figure 2b. The critical points occur at 0, 1, 2, 5, 13, 21, and 50.

To find the critical points, we make four conjectures about the form of function $(tx \bmod c_2)$. The resultant expression has been successfully compared to the numerical evaluation of critical points by testing all values of $x \in (0, c_2]$ in (5) at numerous values of c_1 and c_2.[9]

Conjecture 1. *In a graph of* i *versus* x, *the critical points lie on one of a sequence of* S *line segments, where* S *is a positive integer. If* $S > 1$, *each line segment* $s \in [1, 2, \ldots, S - 1]$ *shares an end point with line segment* $s + 1$.

- *One end point of segment 1 is not shared with any other segment. It is the point* $x = 0$, $i = c_2$
- *One end point of segment* S *is not shared with any other segment. It is the point* $x = c_2$, $i = 0$.

Example. Consider the graph of equation (5), in Figure 2b. Its critical points ($x = 0, 1, 2, 5, 13, 21, 50$) lie on $S = 4$ line segments, as illustrated in Figure 4. Points $x = 0, i = 50$ and $x = 2, i = 12$ are the end points of segment $s = 1$, and points $x = 2, i = 12$ and $x = 5, i = 5$ are the end points of segment $s = 2$.

Conjecture 2. *On any single line segment, the difference between the* x-coordinates *of any two consecutive critical points is the same.*

The two conjectures suggest describing the critical points along a single line by three integers m_s, n_s, and e_s. The x-axis distance between two consecutive critical points is denoted m_s. We denote by $n_s + 1$ the number of

[9] Specifically, we tested all integer combinations of c_1 in the range $[1, 100]$ and all c_2 in the range $[3, 100]$. Then we tested all integer combinations of c_1 in the range of $[1000, 1100]$ and all c_2 in the range of $[1300, 1400]$. Finally, we tested about 50 random points, for which we constructed and studied graphs of i versus x in the process of formulating our conjectures. These random values were in three categories: $c_1, c_2 \in [1, 50]$, $c_1, c_2 \in [100, 500]$, and $c_1, c_2 \in [10000, 11000]$.

critical points along line segment s.[10] Finally, the x coordinate of the rightmost end point on line segment s is denoted e_s.

Example. Returning to Figure 4, the rightmost end points are $e_1 = 2$, $e_2 = 5$, $e_3 = 21$, and $e_4 = 50$. The number of critical points on each segment is $n_1 + 1 = 3$, $n_2 + 1 = 2$, $n_3 + 1 = 3$, and $n_4 + 1 = 2$. Finally, the distance between successive critical points on the first line segment $(x = 0,1,2)$ is $m_1 = 1$, and on the remaining line segments $m_2 = 1$, $m_3 = 8$, and $m_4 = 29$.

If we knew how to calculate m_s, n_s, and e_s for $s \in [1, S]$, then we could find all critical points and hence function w(I). The remaining two conjectures describe how to overlay a graph of (4) with a grid to calculate m_s, n_s, and e_s.

Conjecture 3. *It is possible to draw two sets of parallel lines on a graph of* I *versus* x *such that the intersection points are exactly all points in the graph of equation (5). One set of lines has positive slope, and the second set negative slope. Further, the parallel lines of each set are drawn with equal spacing between them.*

Example. The parallel lines of Conjecture 3 form a grid as illustrated for the case of $c_1 = 29$, $c_2 = 50$ in Figure 5. The parallel lines in each of the two sets are assigned consecutive integer numbers, denoted *u* and *v*, as illustrated in Figure 5.

Conjecture 4. *If we know the critical points along segment* s *(for* $1 \leq s < S$*), we can find the slope of line segment* $s+1$ *by drawing the grid of*

[10] The critical point expression given at the end of this section is simplified by choosing $n_s + 1$, rather than n_s, to represent the number of critical points along segment s.

Conjecture 3 in the following manner: The lines of negative slope are drawn with the line $u=1$ containing line segment s. The lines of positive slope are drawn with the line $v=0$ passing through the origin and through the rightmost end point of line segment s. The slope of line segment $s+1$ is determined by two points:

(a) One point is the intersection of lines $v=0$ and $u=1$

(b) The other point is the intersection of $v=1$ and the leftmost intersection of $v=1$ with any line u.

Example. Figure 5 illustrates the application of Conjectures 3 and 4 to find the critical points along segment $s+1=2$. The lines of negative slope are drawn parallel to line segment $s=1$. The lines of positive slope are drawn with the line $v=0$ passing through the origin and the rightmost end point of segment $s=1$, which is $x=2$, $i=12$. This point is also one end point of line segment 2, by A above. Further, by B above, another point on segment 2 lies on the line $v=1$. In particular, it lies at the intersection of $v=1$ with $u=2$, because there is no line with a smaller value of u that intersects line $v=1$. Thus, points $x=2$, $i=12$ and $x=5$, $i=5$ determine the slope of line segment 2. these points are indicated by the two shaded black boxes in Figure 5.

Using Conjectures 1 and 4, we can obtain the number of critical points along line segment s (i.e., n_s+1) and hence the rightmost end point e_s in the following manner. Conjectures 4 (a) and (b) describe how to find the two leftmost points on line segment s, and hence its slope. We can draw a line

λ through the points specified in Conjectures 4 (a) and (b); thus it contains line segment s. By elementary geometry, we calculate where λ intersects the x-axis. From Conjecture 2, we know that the distance between consecutive critical points along line s is the constant m_s. Thus, we can calculate the maximum number of points which will fit on the portion of λ the line through segment s above the x-axis, which is $n_s + 1$.

Example. In Figure 5, we can calculate $n_2 + 1$ as follows. Line segment 2 has slope $-7/3$. Thus, a line λ containing segment 2 has equation $l = 50/3 - 7/3x$ (see Figure 6). Line λ intersects the x-axis at $x = 7\ 1/7$. The two critical points known from Conjectures 4 (a) and (b) occur at $x = 2$ and $x = 5$; therefore by conjecture 2, $m_2 = 3$. If there were more critical points on segment 2, they would have to occur at $x = 8, 11, 14, \ldots$ However, because λ intersects the x-axis at $x = 7\ 1/4$, no critical points can have a value larger than 7 1/4.

To summarize, the general procedure for calculating the critical points presented in this section is:

(1) Set $s = 1$. Calculate the end points of line segment 1

(2) Increment s

(3) Use Conjecture 4 to calculate two points on line segment $s + 1$

Set $m_s + 1$ to the difference between the x coordinates of the two points (Figure 5)

(4) Based on the previous step, we calculate the rightmost end point of line segment $s + 1$ (i.e., e_{s+1}) and the number of critical points on line

segment $s + 1$ (i.e., $n_s + 1$) (Figure 6)

(5) Steps 2 through 4 are repeated until we generate c_2 as the rightmost end point of some line segment (i.e., $e_s = c_2$). The number of segments, S, is set to the current value of s.

This procedure yields the expression shown below. ıhe derivation is discussed in the Appendix.

Critical-point expression. Consider a two-process distributed program in which process 1 unilaterally shares a resource with process 2. Let $f(x) = tx \bmod c_2$, $s \in [1, S], k \in [1, n_s]$. At steady state, the resource waiting time $w(l)$ will be

$$w(l) = e_{s-1} + m_s k \quad ,$$

where:

$$m_s = \begin{cases} 1 & \text{if } s = 1 \quad (6a) \\ e_{s-1} \left\lceil \dfrac{f(e_{s-2}) - f(e_{s-1})(1 + n_{s-1})}{n_{s-1}\, f(e_{s-1})} \right\rceil + m_{s-1} & \text{otherwise} \quad (6b) \end{cases}$$

$$n_s = \begin{cases} \left\lfloor \dfrac{c_2}{c_2 - t} \right\rfloor & \text{if } s = 1 \quad (7a) \\ \left\lfloor \dfrac{e_{s-1}\, f(e_{s-1})\, n_{s-1}}{e_{s-1}\, f(e_{s-2}) - e_{s-2}\, f(e_{s-1}) - m_s\, n_{s-1}\, f(e_{s-1})} \right\rfloor & \text{otherwise} \quad (7b) \end{cases}$$

$$e_s = \sum_{k=1}^{s} n_k\, m_k \tag{8}$$

$$S \text{ satisfies } e_S = c_2 \tag{9}$$

$$f(e_0) \text{ is defined to be } c_2$$

and s and k are integers chosen to satisfy

$$l \in \begin{cases} [e_{s-1} + m_s k,\ e_s) & \text{if } k = n_s \\ [e_{s-1} + m_s k,\ e_{s-1} + m_s(k+1)) & \text{if } k < n_s \end{cases} .$$

6. IMPLICATIONS OF ANALYSIS

We obtained two results. First, we showed that all programs fitting our model reach a cyclic steady state. Next, we derived an expression for $w(l)$, the waiting time of process 1 as a function of how long process 2 uses the resource. As illustrated in Figure 3, qualitatively $w(l)$ is a step function, with the width of each step increasing with l. These results have the following implications:

(1) The waiting time $w(l)$ is monotonically increasing. Therefore, a program should have *short* resource-use durations to *minimize* the waiting time. We proved that the limiting case of *zero* waiting time occurs when the resource-use duration is less than the greatest common divisor of the cycle lengths of the two processes.

(2) We define a program as being *unstable* if a small change in the resource use duration causes the waiting time to cross a discontinuity, thereby sharply increasing or decreasing. (For example, a program operating at $l = 3$ in Figure 3 is unstable.) The space between discontinuities in $w(l)$ grows as l grows. This implies that a program is

stabler if its resource-use duration is *long*. With increasing I, the likelihood that the program is near a discontinuity decreases.

(3) No waiting occurs during the transient period, before the program enters steady state.[11] If the program only needs to run for a finite period of time, we can try to make the transient as long as possible, by *decreasing I*, to avoid contention. The transient duration increases to a maximum of $c_1 \cdot c_2$ as I shrinks to the g.c.d. of c_1 and c_2.[12]

This work has a broader implication for modeling deterministic distributed programs in general: the program displays potentially unpredictable behavior for small changes in parameters. This contrasts with continuous systems, such as electrical circuits modeled by a differential equation. A nice property of continuous systems is that whenever a parameter of the system is changed slightly, the output behavior changes in a smooth, *predictable* manner.

For example, consider the following enumeration of critical points using the expression developed for w(I):

[11] Processes enter steady state the first time they contend for the resource; therefore no resource contention occurs during the transient.

[12] Graphs of equation (5) of i versus x in Figure 2 illustrate that the number of loops, i, assumes values corresponding to the successive minimums of $f(x) = Ix \bmod c_2$. Thus a small value of I assures that $x \in (0,I]$ will correspond to a large value of i. The maximum transient period is, by (5), c_2 loops of process 1. Because process 1 makes one loop in time c_1, the maximum transient duration is time $c_1 \cdot c_2$.

c_1	c_2	critical values of I at which steady-state cycle changes							
77	90	1	13						
77	91	7	14						
77	92	1	3	5	7	9	11	13	15
77	93	1	2	3	16				

If a programmer is running the program with $c_1 = 77$ and $c_2 = 91$, then, depending on whether the value of I chosen lies in the region $I \in [0,7)$, $[7,14)$, or $[14,90)$, one of *three* steady-state behaviors will appear. But if c_2 is increased by just 1, suddenly *nine* regions of I will emerge, corresponding to nine alternate steady-state behaviors. The region $[0,7)$ is now split into four regions, and region $[7,14)$ is divided into three regions. If c_2 is again increased by one, the number of regions will shrink to five.

The erratic change in number of regions suggests how complex the microscopic behavior underlying some distributed systems may be. This raises the question of whether distributed programs with more processes and resources are more or less erratic. Experimental results in [1] suggest that aggregation of many microscopic discontinuities blends into a system which macroscopically displays relatively few points of discontinuity. This is encouraging for further analysis.

7 APPENDIX

Summarized below is the erivation of expressions for the distance between critical points (m_s) and the number of critical points (n_{s+1}) for each line segment s. The complete derivation appears in [1]. First, we shall consider line segment $s = 1$ and derive (6a) and (7a). Later, we shall derive (6b) and (7b), for $s > 1$, by carrying out the five-step procedure given in Section 5.

A.1. Critical Points on Line Segment s = 1

In Conjecture 1, we defined one end point of line segment $s = 1$ to be the point $x = 0$, $i = c_2$. By definition $x = 1$, $i = t$ is a critical point. Thus, the slope of line 1 is $-(c_2 - t)/1$, and $m_1 = 1$. Now we want to know how many critical points $x = 2, 3, \ldots, n_1$ also lie on this line.

One way to view function $f(x) = tx \bmod c_2$ is as a graph of tx "folded over" on itself every c_2 units (Figure 7). Horizontal lines are drawn in the figure at integral multiples of c_2. The values of f(x) may be derived graphically using Figure 7 by calculating, for each x, the distance from tx to the previous multiple of c_2. We can show that

$$\left\lfloor \frac{t(x+1)}{c_2} \right\rfloor = \begin{cases} \left\lfloor \frac{tx}{c_2} \right\rfloor + 1 & \text{if } x \text{ is a critical point} \\ \left\lfloor \frac{tx}{c_2} \right\rfloor & \text{otherwise} \end{cases} \tag{A1}$$

In terms of Figure 7, the point $x = 2$ is critical, while 3 is not, because $\lfloor 7 \cdot 3/11 \rfloor = \lfloor 7 \cdot 2/11 \rfloor$. This gives us a condition for finding the smallest critical points: the points 1,2,...,n sub 1 are critical, where $n_1 + 1$ is the smallest integer for which $\lfloor t(n_1 + 1)/c_2 \rfloor = \lfloor tn_1/c_2 \rfloor$. Because (4) is equivalent to (5), $f(x) = tx - c_2 \lfloor tx/c_2 \rfloor$. Thus, at $x = n_1 + 1$

$$f(x) = t(n_1 + 1) - c_2 \left\lfloor t \frac{(n_2 + 1)}{c_2} \right\rfloor \ .$$

Substituting (A1) yields

$$f(x) = t(n_1 + 1) - c_2(n_1 - 1) \ . \tag{A2}$$

Because $0 \leq f(x) < c_2$, substituting (A2) for f(x) and solving for n_1 yields (7a).

A.2. Critical Points on Line Segments $s \geq 1$

Now we proceed to the case of $s > 1$. First, we obtain an invertible relation between $f(x) = tx \bmod c_2$ and parameters u and v

$$\begin{aligned} f(x) &= a_f u + b_f v \\ x &= a_x u + b_x v \ . \end{aligned} \tag{A3}$$

To determine parameters a_f, b_f, a_x, and b_x, we need two points, other than the origin, for which the u and v parameters are known. Lines $u = 1$ and $v = 0$ always pass through the critical point at $x = e_{s-1}$:

$$\begin{aligned} f(e_{s-1}) &= a_f \cdot 1 + b_f \cdot 0 \\ e_{s-1} &= a_w \cdot 1 + b_w \cdot 0 \ . \end{aligned} \qquad \text{(A4)}$$

The lines $u = 1$ and $v = -n_{s-1}$ always pass through the critical point at $x = e_{s-2}$:

$$\begin{aligned} f(e_{s-2}) &= a_f \cdot 1 - b_f n_{s-1} \\ e_{s-2} &= a_x \cdot 1 + b_x \cdot 0 \ . \end{aligned} \qquad \text{(A5)}$$

Solving (A4) and (A5) for a_f, b_f, a_x, and b_x, and then substituting into (A3) yields the desired relation

$$f(x) = f(e_{s-1})u + \frac{f(e_{s-1}) - f(e_{s-2})}{n_{s-1}}v \qquad \text{(A6a)}$$

$$x \ = e_{s-1}u + m_{s-1}v \ . \qquad \text{(A6b)}$$

Using (A6) with the conjectures, we shall find expressions for m_s and n_s. First, we obtain two functions which simplify the derivation: $u(v)$ and $x(v)$.

Suppose we wish to find the critical points along line segment s. Each critical point lies on lines $v = 1,2,\ldots,n_s$, at the point with the minimum $f(x)$ value. Function $u(v)$ is defined to be the value of u such that the intersection of v and $u(v)$ is the point with the minimum $f(x)$ value along line v. We can find $u(v)$ from the condition that $f(x) \geq 0$. We substitute (A6a) for $f(x)$, then solve the inequality for $u(v)$. The minimum integer satisfying the inequality is

$$u(v) = \left\lceil \frac{f(e_{s-2}) - f(e_{s-1})}{n_{s-1} f(e_{s-1})} v \right\rceil . \tag{A7}$$

Function $x(v)$, when given a value v on which a critical point lies, yields the x coordinate of that critical point. This function is (A6b) with function $u(v)$ substituted for u:

$$x(v) = e_{s-1} \left\lceil \frac{f(e_{s-2}) - f(e_{s-1})}{n_{s-1} f(e_{s-1})} v \right\rceil + m_{s-1} v .$$

Next, we obtain the expression for m_s, the spacing between two critical points on the same line segment, by subtracting the x coordinates of two adjacent critical points on line segment s. Each line segment contains a critical point at its leftmost end point ($x = e_{s-1}$). Further, the line $v = 1$ must pass through another critical point. Function $x(v)$, evaluated at $v = 1$, yields the x coordinate of this point. Thus, the difference between two critical points on line s is

$$m_s = x(1) - e_{s-1} .$$

Substituting (A7) for x(v) and simplifying yields

$$m_s = e_{s-1}\left[\left\lceil \frac{f(e_{s-2}) - f(e_{s-1})}{n_{s-1}f(e_{s-1})} \right\rceil - 1 \right] + m_{s-1}v$$

Finally, we find an expression for n_s. Each line segment containing critical points has a negative slope. Thus, if we fix v and choose u too large in (A6a), then a negative value of $f(x)$, as defined in (A6a), results. Thus we can find the number of critical points along line s by finding the largest integer n_s for which $f(x) \geq 0$. Substituting (A6a) yields

$$f(e_{s-1})u(n_s) + \frac{f(e_{s-1}) - f(e_{s-2})}{n_{s-1}}n_s \geq 0 \tag{A8}$$

We can eliminate $u(n_s)$ by solving (A6b) for u. Further, the x coordinate of the rightmost critical point along line s must be greater by $n_s m_s$ than the leftmost critical point on line s, at $x = e_{s-1}$. Thus,

$$u(n_s) = \frac{n_s m_s + e_{s-1} - m_{s-2}n_s}{e_{s-1}}$$

Substituting the above in (A8) and solving for n_s yields

$$n_s = \left\lfloor \frac{e_{s-1} f(e_{s-1}) n_{s-1}}{f(e_{s-1})[(m_{s-1} - m_s) n_{s-1} - e_{s-1}] + e_{s-1} f(e_{s-2})} \right\rfloor$$

Substituting e_{s-2} for $m_{s-1} n_{s-1} - e_{s-1}$ yields (7b).

ACKNOWLEDGMENTS

The authors wish to thank Satish Tripathi for several discussions during the course of this work, and Liba Svobodova and Michael Mazurek for suggestions on the manuscript

REFERENCES

1. ABRAMS, M. *Performance analysis of unconditionally synchronizing distributed computer programs using the geometric concurrency model*. Tech. Rep. RT−669, Ph.D. Dissertation, Department of Computer Science, Univ. of Maryland, 1986, Ch. 4−5.

2. BELLMAN, R., ESOGBUE, A. O., AND NABESHIMA, I. *Mathematical Aspects of Scheduling and Applications*. Pergamon Press, New York, 1982.

3. COHEN, G., DUBOIS, D., QUADRAT, J. P., AND VIOT, M. A Linear-System-Theoretic View of Discrete-Event Processes and Its Use for Performance Evaluation in Manufacturing. *IEEE Trans. Automat. Contr. AC-30*, 1985, pp. 210-220.

4. JONES, B. W. *The Theory of Numbers*. Rinehart & Company, Inc., New York 1955.

5. SAUER, C. H., AND CHANDY, K. M. *Computer System Performance Modeling*. Prentice-Hall, Englewood Cliffs, N.J., 1981.

6. VERNON, M., ZAHORJAN, J. AND LAZOWSKA, E. D. *A comparison of performance Petri nets and queueing network models*. Tech. Rep.-669, Computer Sciences Department, Univ. of Wisconsin-Madison, Sept. 1986.

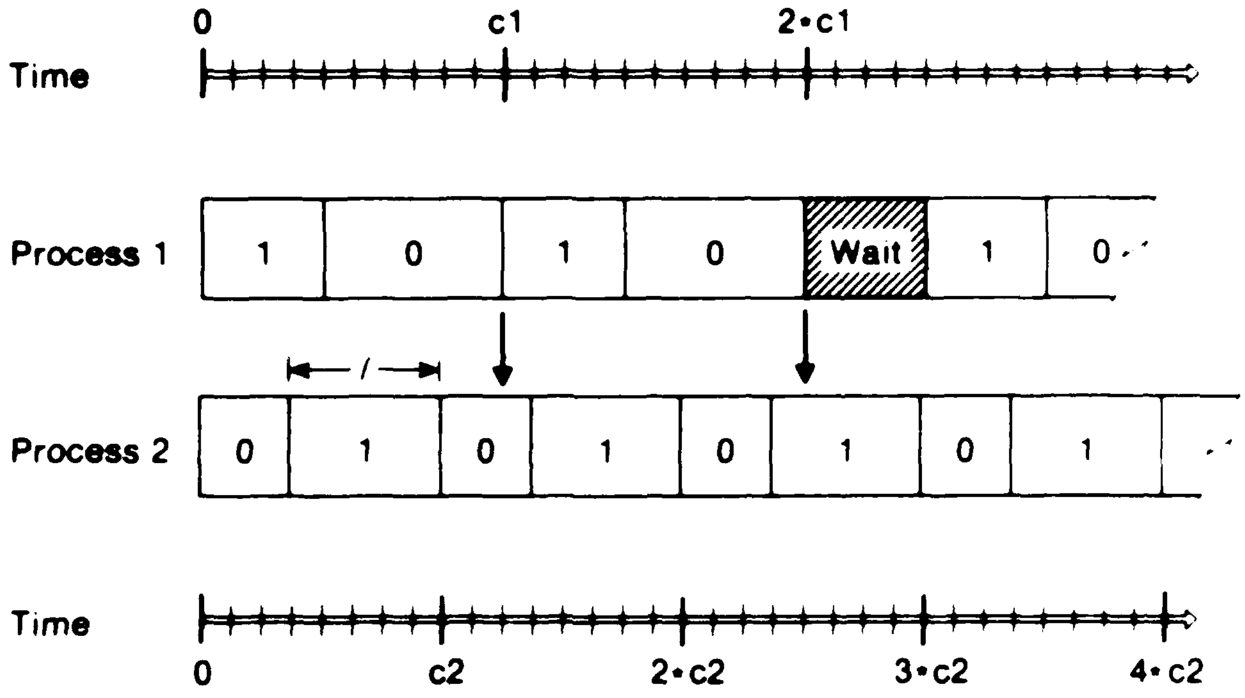


FIG. 1 Typical evolution of program in time after the two processes content for a resource.

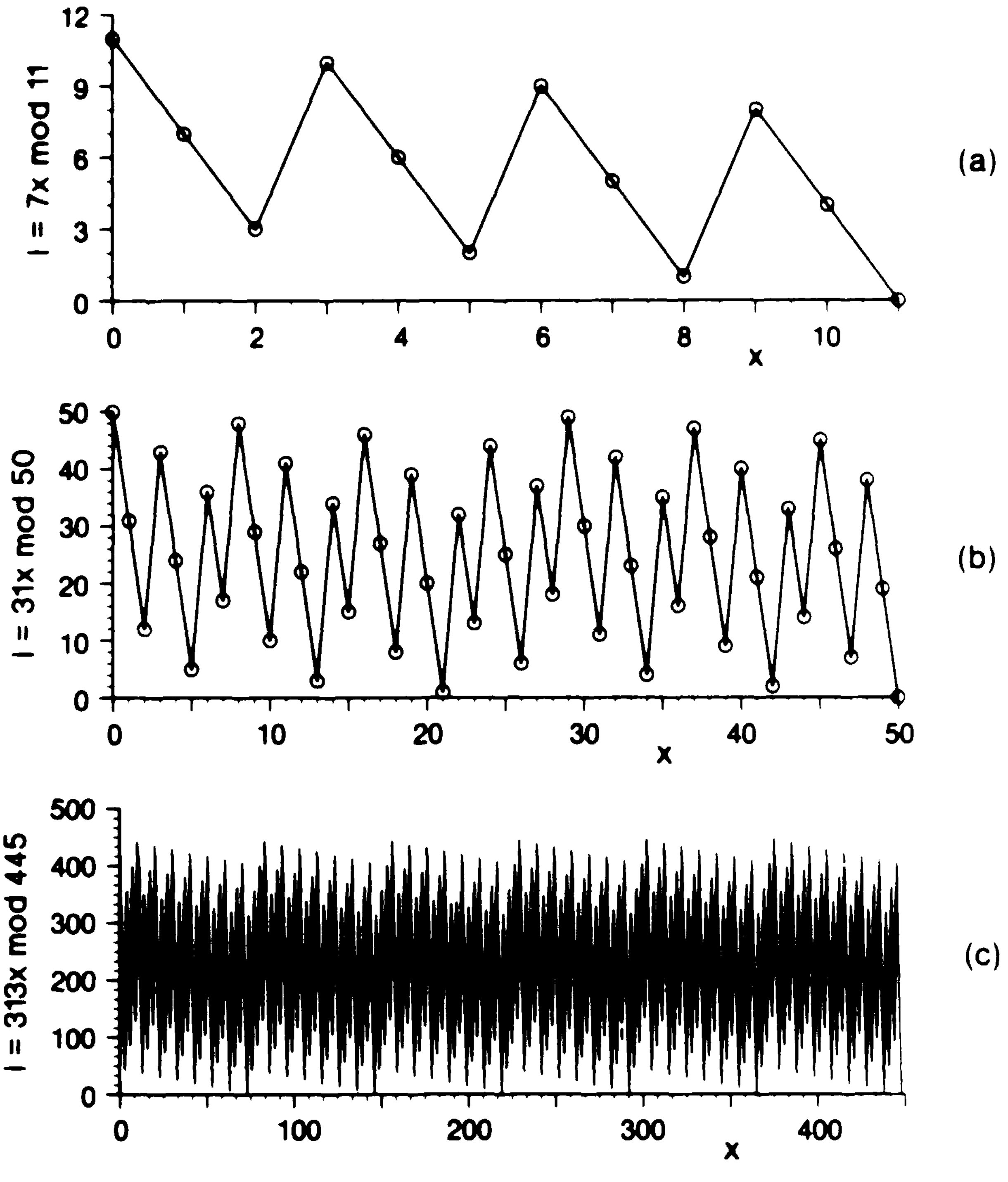


FIG. 2. Three illustrations of equation (5) Graph of function yielding the number of iterations i which process 1 makes between conflicts for resource with process 2.

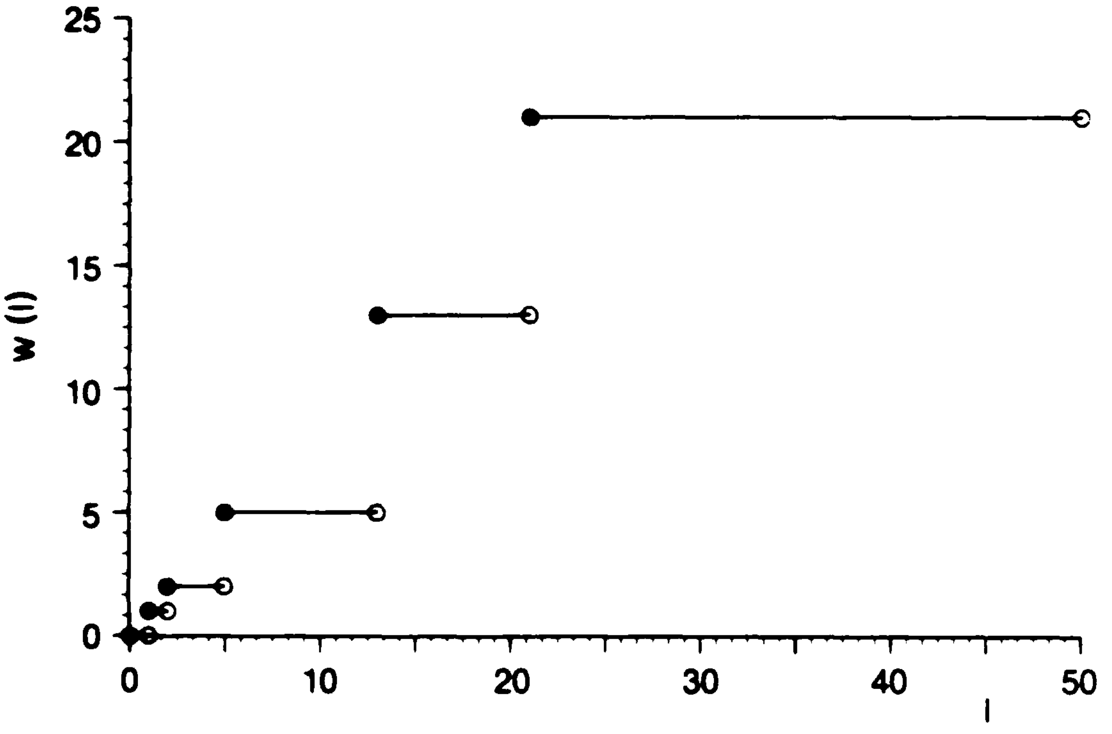


FIG. 3. Graph of w(l) corresponding to Figure 2b. For $l \in [0,1)$, w(l) is by definition zero, because in this region equation (2) has no solution.

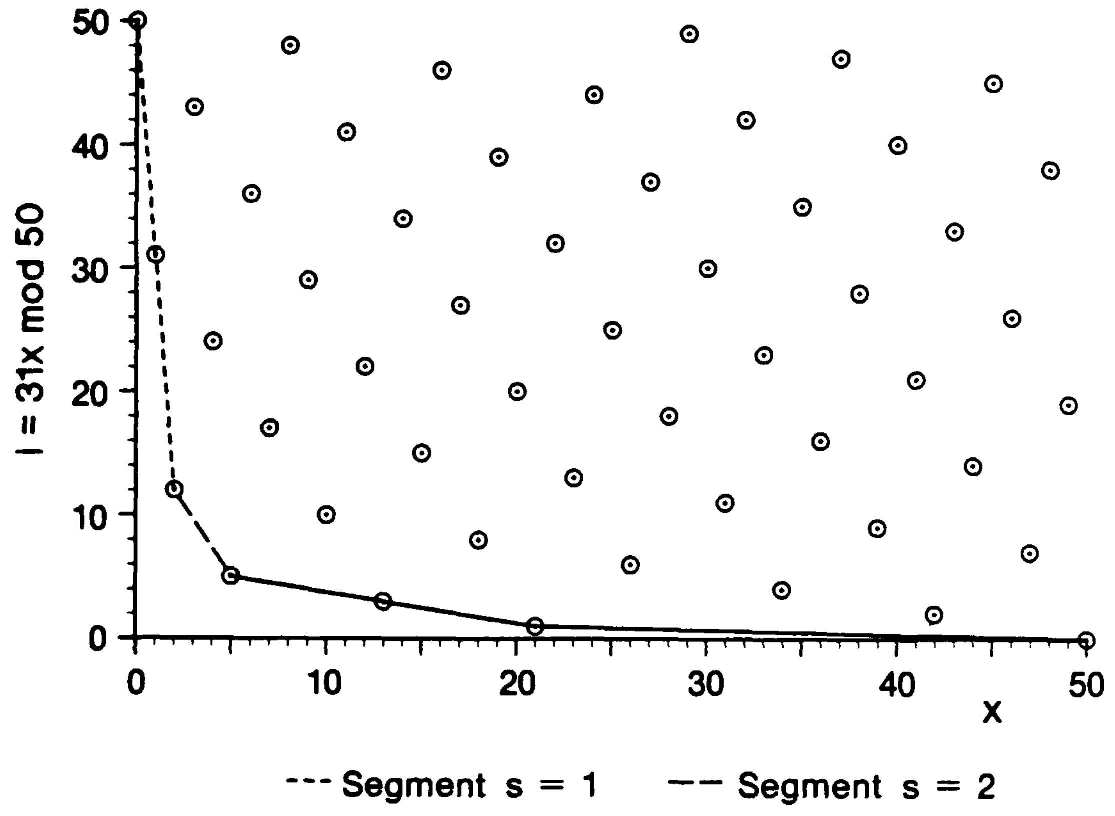


FIG. 4. Illustration of conjecture 1. All critical points lie on one of four line segments.

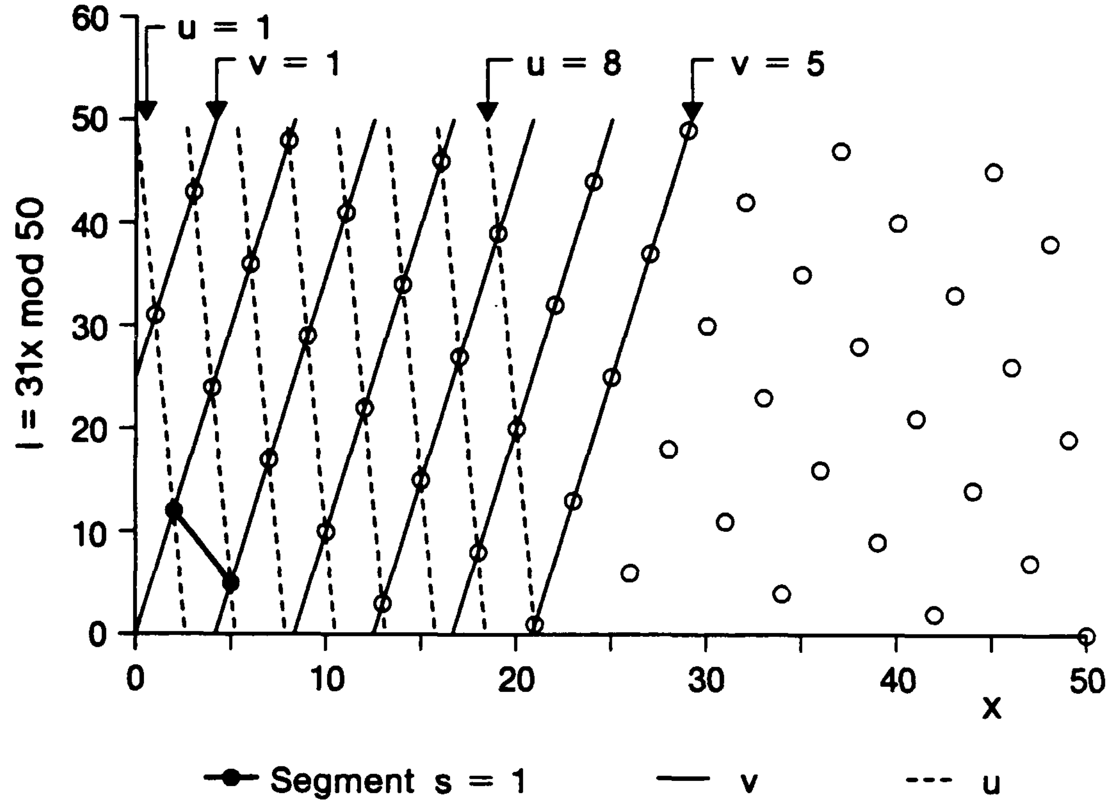


FIG. 5. Illustration of grid of conjectures 3 and 4 overlaying graph of Figure 2b. Grid is oriented to obtain the critical points x = 2 and x = 5 lying on line segment s + 1 = 2. The two black circles represent the two points specified in Conjectures 4 A and B that determine the slope of segment s + 1.

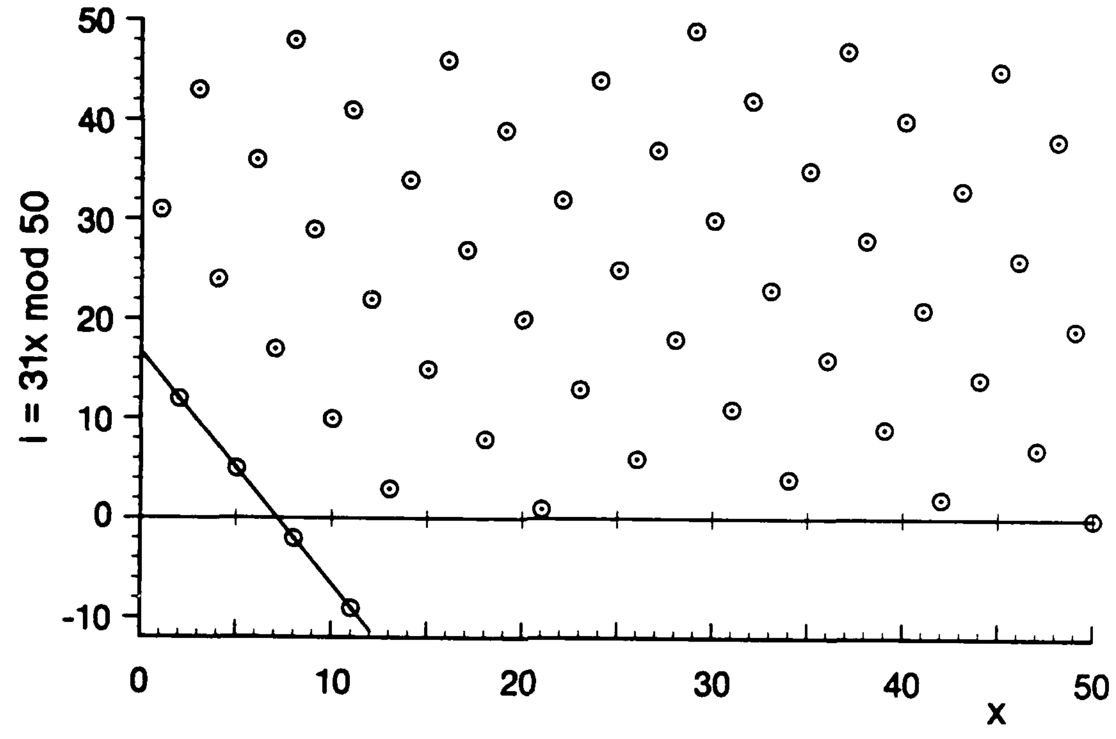


FIG. 6. Illustration of line λ. The line with equation $i = 50/3 - 7/3x$ is used to calculate $n_2 + 1$ for the graph illustrated in Figure 5.

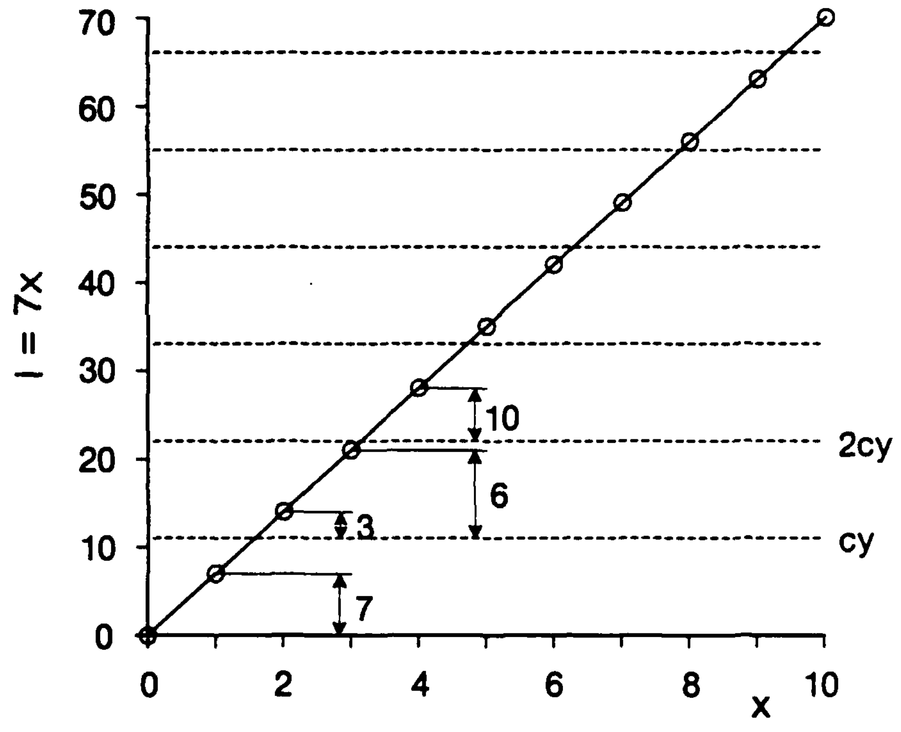


FIG. 7. Function plotted in Figure 2a, when the modulus operation is omitted.

END
8-87
DTIC